CREATIVE & DESIGN TOOLS

Advanced Compositing & Retouching

Photoshop & GIMP, Compared Throughout

Advanced Masking, Compositing & Frequency Separation

Automation, Batch Processing & Plugins

Capstone: Automating a Real Batch-Retouching Workflow

Philip Osztromok

Generated with Claude

Table of Contents

Advanced Compositing & Retouching — 10 Chapters

Raster Editing Fundamentals taught raster editing as a discipline — selections, layers, masks, adjustment layers, the clone/heal tools, basic filters, and exporting for the right destination — with Photoshop and GIMP shown side by side throughout as two independently-built implementations of the same underlying ideas. This course assumes all of that is comfortable working knowledge already, and goes further into the techniques a working professional actually reaches for once the basics stop being the hard part.

A Quick Recap of What Raster Editing Fundamentals Covered

Before going further, it's worth being precise about the floor this course is building on:

- **Selections** — the foundation of nearly every other operation (Chapter 3)
- **Layers** — non-destructive stacking instead of overwriting pixels (Chapter 4)
- **Layer masks** — non-destructive hiding, not deleting (Chapter 5)
- **Adjustment layers & color correction** (Chapter 6)
- **The Clone Stamp & healing tools** for basic retouching (Chapter 7)
- **Basic filters & effects** (Chapter 8)
- **Exporting & file formats** for the right destination (Chapter 9)

If any of that feels shaky rather than second nature, it's worth revisiting those chapters directly before continuing — this course builds on each of them without re-teaching them.

What This Course Adds

Nine chapters follow this one, each pushing one of those fundamentals into genuinely advanced territory:

- **Advanced masking** — channel-based selection for fine detail like hair, well beyond a basic layer mask (Chapter 2)
- **Compositing multiple images seamlessly** — matching color, lighting, and perspective across sources (Chapter 3)
- **Smart Objects & non-destructive filters** — and a real, concrete gap between the two tools here (Chapter 4)
- **Frequency separation** — a genuine professional retouching technique, separating texture from tone (Chapter 5)
- **Automating repetitive work** — Actions and Script-Fu, two genuinely different scripting paradigms (Chapter 6)
- **Batch processing** — running one operation across hundreds of files unattended (Chapter 7)
- **Advanced filters & effect stacking** (Chapter 8)
- **Plugins & extensions** — including a real, practical cost difference between the two tools (Chapter 9)
- **Capstone** — every technique above combined into one production-style batch-retouching workflow (Chapter 10)

RASTER EDITING FUNDAMENTALS	ADVANCED COMPOSITING & RETOUCHING
One selection at a time, by hand	**Channel-based selection for fine, hard-to-select detail**
Editing a single image	**Compositing several images into one believable scene**
Adjustment layers, applied manually per image	**Smart Objects & non-destructive filters, editable after the fact**
Clone Stamp / Healing Brush, by eye	**Frequency separation — texture and tone edited independently**
Every edit performed by hand	**Actions / Script-Fu automating repeated steps**
One file at a time	**Hundreds of files, unattended**

WHY THIS SPLIT EXISTS IN REAL STUDIOS, NOT JUST IN THIS COURSE

Professional retouching workflows genuinely separate "basic" work — a quick crop, a levels adjustment, a blemish removed — from "advanced" work like frequency separation or a fifty-image product-photo batch, because the second category has a real time cost that only pays off once it's automated or applied at scale. This course's own split between Fundamentals and Advanced isn't an arbitrary curriculum device — it mirrors how the work actually gets divided in practice.

WHAT THIS COURSE DELIBERATELY DOES NOT COVER

Two things are out of scope on purpose, not by oversight. First, **AI-generative editing** — Photoshop's Generative Fill/Expand and Adobe Firefly integration, or GIMP's own much younger AI-plugin ecosystem — is a fast-moving space evolving independently of the manual compositing/retouching discipline this course teaches, and deserves its own separate treatment rather than a rushed mention here. Second, **print-production concerns** — CMYK color separation, prepress proofing, bleed and trim setup — belong to a print-specific workflow this course doesn't attempt to cover. Both are genuine gaps, named honestly rather than glossed over.

Who This Course Is For

Anyone comfortable with Raster Editing Fundamentals' own ten chapters — whether completed through this site or through equivalent working experience in Photoshop or GIMP. No new software installation is needed beyond what Raster Editing Fundamentals already assumed; every technique here uses tools already present in both applications, just applied with more depth and intention.

Hands-On Exercises

EXERCISE 1

A colleague who finished Raster Editing Fundamentals asks why frequency separation and batch processing weren't just taught as two more chapters of that course, instead of splitting them into a separate "Advanced" course. Using this chapter's own tip box, explain the reasoning.

View solution

EXERCISE 2

List three techniques from Raster Editing Fundamentals that this course assumes are already comfortable, and match each to the specific chapter of this course that builds directly on top of it.

View solution

EXERCISE 3

A client asks whether this course will teach them to use Photoshop's Generative Fill to remove a person from a photo. Using this chapter's own warning box, explain why that request falls outside this course's scope, and what it does cover instead for removing unwanted elements from a photo.

View solution

CHAPTER 1 QUICK REFERENCE

- This course assumes **Raster Editing Fundamentals**' ten chapters (selections, layers, masks, adjustment layers, clone/heal, filters, export) as comfortable working knowledge
- Nine chapters follow: advanced masking, compositing, Smart Objects/non-destructive filters, frequency separation, automation (Actions/Script-Fu), batch processing, filter stacking, plugins, and a capstone
- The Fundamentals/Advanced split mirrors a real distinction in professional workflows, not just a curriculum device
- **Out of scope, on purpose:** AI-generative editing (Generative Fill/Expand) and print-production concerns (CMYK, prepress)
- No new software is required beyond what Raster Editing Fundamentals already assumed

Raster Editing Fundamentals' Chapter 5 taught the layer mask: paint white to reveal, black to hide, on a mask attached to a layer. That works well for clean, simple edges — a straight horizon, a rectangular product shot. It breaks down completely for fine, tangled detail like hair, fur, or a semi-transparent veil, where painting every individual strand by hand is both impossibly slow and visibly imprecise. This chapter covers the professional alternative: building a selection from the image's own contrast, using its color channels, rather than painting one from scratch.

Why a Painted Mask Breaks Down for Fine Detail

A brush-painted mask is only as precise as your hand and your zoom level allow. Hair against a background has dozens or hundreds of individual strands, each a pixel or two wide, often semi-transparent where fine flyaway hairs let the background show through slightly. Painting each one is not just slow — it's the wrong tool for the job entirely, since a brush can't naturally reproduce that kind of soft, irregular, semi-transparent edge no matter how carefully it's used.

The Channels Panel: Using the Image's Own Contrast as a Selection Source

Every RGB image already contains three separate grayscale images — its Red, Green, and Blue channels — and one of them often shows far more contrast between your subject and the background than the others. The core technique:

1. Open the Channels panel and inspect the Red, Green, and Blue channels individually to find the one with the strongest contrast between subject and background.
2. **Duplicate** that channel — never edit the original, which is still needed for the final image's actual color.
3. Boost contrast on the duplicate (Levels or Curves) until the subject is close to solid white and the background close to solid black.
4. Paint over any remaining gray areas by hand to clean up the result, then load the duplicated channel as a selection.
5. Use that selection to build a precise layer mask back on the original layer.

> **ALWAYS DUPLICATE THE CHANNEL FIRST**
>
> It's easy to start adjusting contrast directly on the Red, Green, or Blue channel itself instead of a duplicate — and just as easy not to notice, since the canvas still looks the same until you switch back to the full-color composite view. Editing the original channel permanently damages the image's actual color data. Duplicating first costs nothing and makes the mistake impossible.

Photoshop's Select and Mask Workspace

Photoshop packages a faster path to the same result: the **Select and Mask** workspace (accessible from any selection tool's options bar). Its **Refine Edge Brush** lets you brush directly over a hair-like edge, and Photoshop automatically detects fine detail within that brushed area using the same underlying channel-contrast logic described above — without manually building a duplicate channel yourself. Its **Decontaminate Colors** option additionally removes leftover background color fringing from semi-transparent edge pixels, replacing it with a plausible extension of the subject's own color.

GIMP's Channel-Based Approach

GIMP has no single packaged workspace equivalent to Select and Mask — the manual channel technique above *is* GIMP's primary approach for this kind of mask, done directly through its own Channels dialog, Curves, and the Threshold tool. GIMP's **Foreground Select tool** gets partway to Photoshop's automation for simpler cases (marking foreground and background roughly, then refining an initial matte), but for genuinely fine detail like hair, GIMP users lean on the manual channel workflow far more than Photoshop users typically need to.

ASPECT	PHOTOSHOP	GIMP
Packaged workflow	Select and Mask workspace, with a dedicated Refine Edge Brush	No dedicated workspace — the manual channel technique is the primary method
Edge fringing fix	Decontaminate Colors, built in	No direct equivalent — handled manually via additional masking/painting
Underlying technique	Same channel-contrast logic, automated	Same channel-contrast logic, performed by hand
Simpler-case shortcut	Quick Selection + Select and Mask	Foreground Select tool

CHANNEL MASKING NEEDS CONTRAST TO WORK WITH

This technique depends entirely on there being real contrast, in at least one channel, between subject and background. A subject photographed against a similarly-colored, busy, or low-contrast background won't yield a clean result no matter how carefully the channel is adjusted — there's simply no separating signal to extract. This is one of several reasons product and portrait photographers shoot against a plain, contrasting backdrop whenever a clean extraction is planned. Chapter 3's own compositing work assumes a usably clean mask already exists — a poor source photo is a limitation no amount of masking skill can fully overcome.

Hands-On Exercises

EXERCISE 1

Explain, step by step, why you should duplicate a color channel before adjusting its contrast for a mask, and what specifically goes wrong if you skip that step.

View solution

EXERCISE 2

A GIMP user tells a Photoshop user "Decontaminate Colors sounds like magic — GIMP must be missing something important." Using this chapter's own material, explain what Decontaminate Colors actually does, and what a GIMP user would need to do manually to approximate the same result.

View solution

EXERCISE 3

A client sends a photo of a subject with wild, flyaway hair shot against a cluttered, similarly-colored background, and asks for a perfect hair-detail cutout. Using this chapter's own warning box, explain why this request may not be fully achievable through masking skill alone, and what you'd tell the client.

View solution

CHAPTER 2 QUICK REFERENCE

- A painted layer mask can't reproduce fine, hair-like detail — a channel-based selection can
- Core technique: inspect R/G/B channels for contrast → **duplicate** the best one → boost contrast → load as selection → build a mask
- **Photoshop:** Select and Mask workspace, Refine Edge Brush, Decontaminate Colors (automates the channel technique)
- **GIMP:** the manual channel technique is the primary method; Foreground Select tool helps with simpler cases
- Channel masking needs real contrast to work with — a poor source photo can't be fully rescued by masking skill

Raster Editing Fundamentals' Chapter 7 taught the Clone Stamp and Healing tools — sampling pixels from elsewhere in *the same image* so a fix blends in seamlessly, with no visible seam. Compositing is the same underlying skill — make the seam disappear — applied at a larger scale: instead of blending a patch of sky from one part of a photo into another part of the same photo, you're blending an entire subject from one photograph into an entirely different one. The tool changes, but the goal — a seam nobody can find — is identical.

What Makes a Composite "Read" as Real

A viewer doesn't consciously check color values or shadow angles — but their eye still catches a wrong composite instantly, even if they can't say exactly why. Four things drive that instinct, and a convincing composite needs all four to agree:

- **Color and tone** — do the subject and the new background share the same overall color cast and contrast?
- **Lighting direction** — do shadows on the subject fall the same way shadows fall in the background?
- **Perspective and scale** — is the subject's camera angle and apparent size consistent with where it sits in the new scene?
- **Edge quality** — does the boundary around the subject look like part of the photo, or like a cutout pasted on top of it?

Matching Color and Tone Across Sources

Two source photos shot under different light — a subject under indoor tungsten light, a background shot outdoors at golden hour, for instance — will carry noticeably different color casts even before any obvious mismatch is visible. The adjustment layers taught in Fundamentals Chapter 6 are the exact tool for the job here: sample the background's own overall color balance and contrast, then apply matching Curves or Color Balance adjustments to the subject layer until it plausibly belongs under the same light. Photoshop packages a faster starting point for this with its **Match Color** command, which analyzes a target layer's statistics and nudges a source layer toward them automatically — a useful starting point, though it rarely gets the final result exactly right without manual refinement afterward.

Matching Lighting Direction and Perspective

Color can be corrected numerically, but lighting direction and perspective have to be judged by eye, and mismatches here are usually what a viewer's instinct catches first. Check where the background's own shadows fall, and confirm the subject's own shadows (and the direction their body is lit from) point the same way — a subject lit from the left, dropped into a scene where every shadow falls from the right, will look wrong immediately even to an untrained eye. The same applies to camera angle: a subject photographed from eye level doesn't sit convincingly into a background shot from a high angle looking down, since the two describe a different physical vantage point on the scene.

Feathering and Blending Edges

Chapter 2's channel-based masking produces a precise selection, but a mask edge that's perfectly hard-edged at the pixel level still often looks pasted-on, because real photographic edges are almost never perfectly crisp — there's a soft transition of a pixel or two from focus, motion, or simply the camera's own optical limits. A small amount of **feathering** (softening the mask's edge transition slightly) reintroduces that natural softness. Too little feathering leaves a visible hard edge; too much blurs real detail the mask in Chapter 2 was built specifically to preserve — the right amount is usually only one or two pixels, judged by eye at full zoom.

TASK	PHOTOSHOP	GIMP
Automated color matching	Match Color command	No direct equivalent — manual Curves/Levels matching only
Blending exposure/focus across sources	Auto-Blend Layers	No direct equivalent — manual layer masking
Edge feathering	Feather command / Select and Mask's Feather slider	Select > Feather, same underlying concept
Seamless texture blending (forward reference)	Content-Aware tools	Resynthesizer plugin — covered in Chapter 9

DEPTH OF FIELD AND GRAIN ARE EASY TO FORGET

Color, lighting, and perspective get most of the attention, but a subject shot in crisp focus dropped onto a background with soft, blurred depth-of-field falloff — or a clean digital subject composited onto a grainy film-scanned background — will still look subtly wrong even with everything else matched perfectly. A touch of matching blur or matching grain, added deliberately to the sharper element, is often the final detail that makes a composite stop looking "too clean" against its surroundings.

POST-PROCESSING CAN'T FULLY RESCUE INCOMPATIBLE SOURCE PHOTOS

Color can be adjusted, and a shadow can be painted in convincingly — but a subject photographed from a dramatically different angle than the background, or lit by a fundamentally different light source position that no plausible in-scene light could produce, is fighting a losing battle no matter how much time is spent on it afterward. The most reliable fix isn't a better editing technique — it's planning ahead, shooting (or choosing) source images under reasonably compatible conditions in the first place. This is the same honest limitation Chapter 2 named for masking a poor source photo: skill in this chapter can get you most of the way, but it can't manufacture compatibility that was never there in the source material.

Hands-On Exercises

EXERCISE 1

Explain how the Clone Stamp/Healing skill from Raster Editing Fundamentals Chapter 7 and the compositing skill in this chapter share the same underlying goal, and what specifically changes between the two.

View solution

EXERCISE 2

A composite has correctly matched color and lighting direction, but still looks obviously fake. List two things covered in this chapter besides color and lighting that could be the cause, and explain how each would be fixed.

View solution

EXERCISE 3

A client provides a subject photographed from a low angle looking up, and a background shot from a high angle looking down, and asks for a seamless composite. Using this chapter's own warning box, explain the real limitation here and what you'd recommend instead of attempting the composite as requested.

View solution

CHAPTER 3 QUICK REFERENCE

- Compositing is Chapter 7's "seamless blend" skill (Fundamentals) applied across multiple source images, not just within one
- Four things must agree for a composite to read as real: **color/tone**, **lighting direction**, **perspective/scale**, and **edge quality**
- Match color/tone with adjustment layers (Fundamentals Ch.6); Photoshop's Match Color is a useful starting point, not a finishing tool
- Feather a mask's edge slightly to reintroduce natural photographic softness — too little looks pasted-on, too much destroys detail
- Depth of field and grain mismatches are easy to overlook but often the final giveaway
- Post-processing can't rescue source photos shot under genuinely incompatible angles or light — plan compatible source material first

Raster Editing Fundamentals' Chapter 4 introduced non-destructive layer stacking, and Chapter 6 introduced non-destructive color correction via adjustment layers. Both leave the underlying pixels of the layer below untouched, and both are fully reversible. This chapter covers a container that extends that same non-destructive principle much further: Photoshop's **Smart Object**. This is also the first chapter in this course where Photoshop and GIMP genuinely part ways — not just in interface, but in real capability — and it's worth being honest about that rather than smoothing it over.

What Makes a Smart Object "Smart"

Converting a layer to a Smart Object wraps it in a container that preserves the original pixel (or vector, for a placed illustration) data, completely separately from whatever transforms or filters get applied on top of it. The practical result: scale a normal raster layer down to 20% size, then back up to 100%, and you've permanently lost detail — the pixels that got discarded on the way down aren't coming back. Do the same to a Smart Object, and the scale-down/scale-up round trip loses nothing, because Photoshop always re-renders the transform from the original preserved data, not from whatever the layer currently displays.

Smart Filters

Any filter applied to a Smart Object automatically becomes a **Smart Filter** — listed underneath the layer in the Layers panel, exactly like an adjustment layer's own entry. Each Smart Filter can be double-clicked to reopen with different settings, temporarily disabled, reordered relative to other Smart Filters on the same layer, or given its own layer mask to limit exactly where it applies. None of this requires undoing your way back through history — the filter's settings remain live and editable indefinitely.

Linked vs. Embedded Smart Objects

A Smart Object can be **embedded** (its original data stored inside the PSD file itself, fully self-contained) or **linked** (referencing an external file on disk). A linked Smart Object updates automatically everywhere it's used if the external source file changes — genuinely useful when the same logo or texture appears across many composite files and needs a single point of update, though it does mean the PSD isn't fully self-contained anymore, and moving or renaming the linked file breaks the reference.

GIMP's GEGL-Based Non-Destructive Filters

GIMP 2.10 introduced **GEGL** (Generic Graphics Library) as its underlying image-processing engine, and one real, welcome result is that many GEGL-based filters can be applied with live, adjustable on-canvas settings — reopen the filter dialog and its parameters are still editable, similar in spirit to a Smart Filter. This is genuine progress, not a token gesture.

> **WHERE GIMP'S NON-DESTRUCTIVE WORKFLOW IS GENUINELY MORE LIMITED**
>
> GEGL's non-destructive behavior is real but narrower than a Smart Object in two concrete ways. First, it applies to individual filter settings, not to the layer as a whole the way a Smart Object does — GIMP has no equivalent container that preserves a layer's original pixel data through an arbitrary sequence of transforms (scale, rotate, skew), so scaling a normal GIMP layer down and back up still loses quality exactly the way it would in Photoshop without a Smart Object. Second, not every GIMP filter or tool is GEGL-based or non-destructive — some operations remain immediately destructive once applied, with no equivalent of Photoshop's "reopen and adjust" behavior. This is a genuine capability gap, not just a different way of phrasing the same thing.

ASPECT	PHOTOSHOP (SMART OBJECTS)	GIMP (GEGL)
Non-destructive transforms (scale/rotate/skew)	Yes — original data always preserved	No direct equivalent — transforms remain destructive on normal layers
Non-destructive filter settings	Yes, via Smart Filters	Partial — many GEGL filters, not universally all filters/tools
Filter-specific mask	Yes, per Smart Filter	No direct equivalent — a separate layer mask must be managed manually
Linked external assets	Yes (linked Smart Objects)	No direct equivalent

CONVERT BEFORE YOU TRANSFORM, NOT AFTER

In Photoshop, the habit worth building is converting a layer to a Smart Object *before* scaling or applying a filter you might want to revisit — converting afterward doesn't recover detail already lost to a prior destructive transform. In GIMP, since there's no equivalent container, the practical workaround for a layer you might need to rescale later is keeping an untouched duplicate of the original layer, hidden but preserved, so a mistake can be corrected by returning to that duplicate rather than to a resized copy.

Hands-On Exercises

EXERCISE 1

Explain what specifically happens to image quality when a normal raster layer is scaled down and then back up to its original size, and why a Smart Object avoids that problem.

View solution

EXERCISE 2

A GIMP user says "GEGL means GIMP has non-destructive filters now, so it's fully caught up with Photoshop's Smart Objects." Using this chapter's own warning box, explain what's incomplete about this claim.

View solution

EXERCISE 3

A logo appears in twelve different composite files, and the client asks for its color to be updated everywhere at once. Explain how a linked Smart Object would solve this in Photoshop, and what the practical downside of using one is.

View solution

CHAPTER 4 QUICK REFERENCE

- A **Smart Object** preserves a layer's original data through any transform — scale down and back up with zero quality loss
- A **Smart Filter** is any filter applied to a Smart Object — reopenable, reorderable, maskable, disable-able at any time
- **Linked** Smart Objects reference an external file and update everywhere automatically; **embedded** ones are self-contained
- GIMP's **GEGL** engine gives many filters live, adjustable settings — genuine but narrower than a Smart Object
- GIMP has **no equivalent** for non-destructive transforms or linked assets — a real capability gap, not just a UI difference
- Convert to a Smart Object *before* transforming in Photoshop; keep an untouched duplicate layer as GIMP's manual workaround

This course's own Chapter 1 named frequency separation as the professional evolution of Raster Editing Fundamentals' Clone Stamp and Healing tools — the same underlying goal (smooth an imperfection, keep it looking real) approached with more precision. This chapter covers the actual technique in full: separating a single image into two independent layers, one holding color and tone, one holding fine texture, so each can be retouched without disturbing the other.

The Core Idea

A photographed skin (or any richly textured) surface carries two kinds of information mixed together: **tone** — the broad, slow-changing color and lighting across the surface (blotchiness, shadows, uneven color) — and **texture** — the fine, fast-changing detail (pores, fine wrinkles, fabric weave). A single Clone Stamp or Healing Brush pass edits both at once, which is exactly why heavy-handed retouching often produces the tell-tale "plastic," over-smoothed look: it can't fix uneven tone without also softening away the fine texture that made the surface look real. Frequency separation splits the two apart into independent layers, so tone can be smoothed while texture stays completely untouched, and vice versa.

Building the Two Layers

1. Duplicate the base image layer twice, producing two identical copies stacked on top of the original.
2. On the first duplicate, apply a strong Gaussian Blur — enough to erase fine texture entirely, leaving only broad tone and color. This becomes the **Low Frequency** layer.
3. On the second duplicate, mathematically subtract the blurred Low Frequency layer from it, leaving only what the blur removed — the fine texture detail. This becomes the **High Frequency** layer, typically rendered as a mid-gray image where brighter or darker deviations from gray represent texture detail.
4. Set the High Frequency layer's blend mode to **Linear Light**. Viewed together, the Low and High layers reconstitute the original image exactly — this is a genuinely reversible split, not an approximation.

Retouching Each Layer

With the split built, retouching becomes deliberately layer-specific:

- **On the Low Frequency layer** — smooth blotchy or uneven tone and color with a soft brush, the Clone Stamp, or the Healing Brush, working at low opacity. Because this layer holds no fine texture at all, there's nothing to accidentally soften away.
- **On the High Frequency layer** — remove or duplicate texture-level imperfections (a scar, a deep wrinkle, a blemish's raised texture) using the Clone Stamp or Healing Brush set to sample and blend using Linear Light, working directly on the texture data without touching the tone underneath at all.

Photoshop's Workflow: Apply Image

Photoshop builds the High Frequency layer precisely via the **Apply Image** dialog, which subtracts the blurred layer from the original using an Add blend mode with a Scale of 2 and an Offset of 128 — the specific formula that produces a correctly mid-gray-centered high-frequency layer, reconstitutable exactly via Linear Light. Many retouchers automate the whole multi-step setup with a saved Action (Chapter 6 covers building one).

GIMP's Approach: Wavelet Decompose

GIMP has no single built-in dialog equivalent to Apply Image for this specific purpose. In practice, GIMP retouchers rely on the community-built **Wavelet Decompose** script (a Script-Fu script, not a stock GIMP feature) to automate building the frequency layers — and it typically produces several scale layers rather than just two, letting you target very fine texture separately from medium-scale tone variation. The honest caveat: this script isn't part of a default GIMP install and has to be installed separately, which this course's own Chapter 9 covers directly — a real, practical difference from Photoshop's built-in Apply Image.

ASPECT	PHOTOSHOP	GIMP
Building the split	Apply Image dialog, built in	Wavelet Decompose script — requires separate installation
Number of frequency layers	Typically 2 (Low/High)	Often several scale layers
Reconstitution blend mode	Linear Light	Same underlying math, same blend mode concept
Retouching tools used	Clone Stamp / Healing Brush, per layer	Clone / Heal tool, per layer — same principle

MATCH THE BLUR RADIUS TO THE TEXTURE SCALE YOU'RE SEPARATING

The Gaussian Blur radius used to build the Low Frequency layer isn't arbitrary — too small a radius leaves some real texture bleeding into the Low layer (making it impossible to smooth tone without also softening texture there), while too large a radius blurs genuine tonal transitions (like a real shadow edge) into the High layer, where they'll be mistaken for texture and can get accidentally erased. A useful starting point is a radius roughly matching the physical size of the finest texture detail you actually want preserved — pore-level detail on a portrait needs a much smaller radius than the weave of a coarse fabric.

FREQUENCY SEPARATION IS POWERFUL, AND EASY TO OVERDO

Smoothing the Low Frequency layer too aggressively — even though texture is technically untouched on the High layer — still produces an unnatural, overly even result, because real skin and most real surfaces have some genuine tonal variation that's part of what makes them look alive, not just "imperfections" to erase. This is precisely the over-retouched "plastic" look frequency separation exists to avoid, achieved by the exact same overuse of the exact same technique. It's also a heavier technique than most retouching jobs actually need — reserve it for cases (professional portrait or beauty retouching) where genuinely preserving texture while correcting tone matters, not as a default first step for every photo.

Hands-On Exercises

EXERCISE 1

Explain, in your own words, why a single Clone Stamp pass on an unseparated image struggles to fix uneven skin tone without also softening away real texture — and how splitting the image into Low and High Frequency layers solves that specific problem.

View solution

EXERCISE 2

Explain what goes wrong if the Gaussian Blur radius used to build the Low Frequency layer is set far too large for the texture you're actually trying to preserve.

View solution

EXERCISE 3

A junior retoucher smooths the Low Frequency layer as much as possible, reasoning "the texture layer is untouched, so this can't cause the plastic-skin look." Using this chapter's own warning box, explain what's wrong with that reasoning.

View solution

CHAPTER 5 QUICK REFERENCE

- Frequency separation splits an image into a **Low Frequency** layer (tone/color) and a **High Frequency** layer (texture), so each can be retouched independently
- Build it: duplicate twice → blur one heavily (Low) → subtract Low from the other (High) → set High to **Linear Light** to reconstitute the original
- Retouch tone on the Low layer, texture-level imperfections on the High layer, each without affecting the other
- **Photoshop:** Apply Image dialog, built in. **GIMP:** Wavelet Decompose script — a real gap, requires separate installation (Chapter 9)
- Match the blur radius to the actual texture scale being preserved — too small or too large both cause problems
- Over-smoothing the Low layer still produces the "plastic" look this technique exists to avoid — it's a heavier technique, reserved for jobs that genuinely need it

Every technique so far in this course — building a channel-based mask, setting up frequency separation's two layers, matching color across a composite — involves a repeatable sequence of steps. Once a sequence is proven to work, doing it by hand every single time wastes effort. This chapter covers how Photoshop and GIMP each let you automate that repetition — and unlike every prior chapter's difference, this one is a genuinely different *paradigm*, not just a different menu layout for the same idea.

Photoshop's Actions Panel: Recorded Playback

An **Action** is a recording. Open the Actions panel, hit record, then perform a sequence of steps normally — convert a layer to a Smart Object, apply a filter, flatten, save — and Photoshop records each step as you go. Stop recording, and that exact sequence can be replayed on any other file with a single click, or bound to a function-key shortcut for instant reuse. Actions can include a manual "stop," pausing playback so you can perform one non-automatable step (like judging a crop by eye) before the rest of the recording continues. No programming is involved at any point — you're teaching Photoshop by demonstration, not by writing instructions.

GIMP's Script-Fu: A Real Programming Language

Script-Fu is not a recorder, and this is the genuine paradigm difference worth being precise about: it's a console and interpreter for **Scheme**, a dialect of Lisp, built directly into GIMP. Rather than performing steps once and having them recorded, you write actual code that calls GIMP's own internal function library — its **Procedure Database (PDB)** — directly by name, with real variables, loops, and conditionals available exactly as they would be in any other programming language.

```
; Load an image, flatten it, and refresh the display
(let* ((image (car (gimp-file-load RUN-NONINTERACTIVE
                                   "photo.png" "photo.png")))
(drawable (car (gimp-image-get-active-drawable image))))
(gimp-image-flatten image)
(gimp-displays-flush))
```

Every operation is an explicit function call, wrapped in parentheses, with the function name coming first (prefix notation) — `(gimp-image-flatten image)` means "call gimp-image-flatten, passing image as its argument," not "image dot flatten" the way many other languages would phrase it. This syntax is genuinely unfamiliar to most people coming from any mainstream language, and that unfamiliarity is real, not a minor cosmetic difference.

Why This Difference Matters in Practice

An Action requires zero programming knowledge — anyone who can perform the steps once can create one. Script-Fu requires learning an unfamiliar syntax and genuine programming concepts (variables, conditionals, function calls) before writing anything useful at all. But that investment buys real capability an Action structurally cannot offer: a Script-Fu script can make a decision — "if this image is wider than it is tall, do X; otherwise do Y" — using an actual conditional, while an Action can only ever replay the exact fixed sequence it recorded, with no ability to branch based on what it encounters.

ASPECT	PHOTOSHOP ACTIONS	GIMP SCRIPT-FU
How it's created	Recorded by performing steps once	Written as code, calling PDB functions directly
Underlying paradigm	Macro playback	Real programming language (Scheme)
Conditional logic	Not supported — replays a fixed sequence	Full support — real if/else conditionals
Learning curve	None — demonstrate once, done	Real — unfamiliar prefix-notation syntax
Manual pause mid-sequence	Yes, via a Stop	Possible, but must be coded explicitly

USE THE SCRIPT-FU CONSOLE TO EXPERIMENT BEFORE SAVING A SCRIPT

GIMP's Script-Fu Console (Filters > Script-Fu > Console) evaluates one expression at a time and immediately shows its result — a far friendlier way to learn Scheme's nested-parentheses syntax than writing a full script blind and debugging it only after running the whole thing. Test each PDB function call individually in the console first, confirm it does what you expect, then assemble the working pieces into a saved script afterward.

AN ACTION CAN ONLY REPLAY EXACTLY WHAT IT RECORDED

Because an Action has no real conditional logic, it can behave oddly or fail silently on a file that doesn't match the exact conditions present when it was recorded — a step that assumed a specific starting layer name, image mode, or canvas size can break quietly on a file that differs in some way that wasn't anticipated. This is a genuine structural limitation of recording-based automation, not a minor edge case, and it's exactly the gap Chapter 7's batch processing work has to account for when running either Actions or scripts unattended across many files with real, imperfect variation between them.

Hands-On Exercises

EXERCISE 1

A colleague says "Script-Fu is just GIMP's version of a Photoshop Action, with a different-looking interface." Explain why this understates the actual difference between the two.

View solution

EXERCISE 2

Explain what `(gimp-image-flatten image)` means in terms of ordinary function-call syntax, and why this notation style is called "prefix notation."

View solution

EXERCISE 3

A recorded Photoshop Action was created using a file where the background layer was named "Background," and it now fails silently on a batch of new files where that layer happens to be named something else. Using this chapter's own warning box, explain why this happened and how a Script-Fu script could have avoided the problem.

View solution

CHAPTER 6 QUICK REFERENCE

- **Photoshop Actions** — recorded macro playback; no programming knowledge required; replays a fixed sequence exactly
- **GIMP Script-Fu** — a real Scheme (Lisp) interpreter built into GIMP, calling functions from GIMP's Procedure Database (PDB) directly
- Script-Fu uses **prefix notation** — `(function arg1 arg2)` — genuinely unfamiliar syntax to most newcomers
- Only Script-Fu supports real **conditional logic** — an Action can't branch based on what it encounters
- Test Script-Fu code interactively in the **Script-Fu Console** before saving a full script
- An Action's fixed-sequence replay is a real limitation — it can fail silently on files that don't match its recorded assumptions

Chapter 6 covered creating a Photoshop Action or a GIMP Script-Fu script capable of performing a sequence of steps automatically. This chapter covers the other half of real production automation: running that Action or script against hundreds of files unattended, with no manual per-file intervention at all — the actual task a working retoucher reaches for automation to solve in the first place.

Photoshop's Batch Command

File > Automate > Batch takes a previously-recorded Action and a source folder, and runs that Action against every file in the folder automatically. It offers options for handling anything the Action recorded that needs adapting per run — overriding "Open" and "Save" steps so files read from and write to the correct folders rather than the exact paths that happened to be open while recording, and choosing how to handle a recorded "Stop" (skip it, or still pause for each file). For the specific, extremely common case of resizing and re-saving a batch of files in a new format, Photoshop also offers a more specialized dialog — the **Image Processor** (File > Scripts > Image Processor) — which handles that one task directly without needing to build a custom Action first.

GIMP's Batch Mode

GIMP's batch mechanism works completely differently at a technical level: GIMP can be launched from the command line with a `-b` (batch) flag, passing Script-Fu code directly as a command-line argument, processing files without ever opening its normal graphical window at all. This isn't "the GUI app running against a folder" the way Photoshop's Batch command is — it's GIMP running headless, purely as a processing engine driven entirely by code.

```
# Run GIMP with no GUI, execute a Script-Fu expression, then quit
gimp -i -b '(let* ((image (car (gimp-file-load RUN-NONINTERACTIVE "in.jpg"
"in.jpg"))))
             (gimp-image-scale image 800 600)
             (gimp-image-flatten image)
             (file-jpeg-save RUN-NONINTERACTIVE image
               (car (gimp-image-get-active-drawable image))
               "out.jpg" "out.jpg" 0.9 0 1 1 "" 0 1 0 0))' \
     -b '(gimp-quit 0)'
```

A real production script wraps this same logic in a loop over every file in a source folder, rather than hard-coding one filename as shown above for clarity. GIMP additionally offers **Python-Fu** as a second scripting language option alongside Script-Fu — the same underlying GIMP Procedure Database, but callable from ordinary Python syntax instead of Scheme's prefix notation. It's worth flagging honestly here, since Chapter 6 covered Script-Fu specifically: Python-Fu is a genuinely more approachable entry point for anyone already comfortable with Python, and many real-world GIMP batch scripts are written in it rather than Script-Fu for exactly that reason.

A Real Production Task: Resize & Watermark Hundreds of Images

A common real job: an e-commerce client provides a folder of several hundred raw product photos, each needing resizing to a consistent dimension, a semi-transparent logo watermark composited into a corner, and saving into an output folder under a consistent naming scheme — all without manually opening each file. Both tools solve this the same way in principle: build and test the sequence on one file first (resize, composite watermark layer at reduced opacity, flatten, export), then apply that proven sequence across the whole folder via Photoshop's Batch/Image Processor or a GIMP batch-mode loop.

ASPECT	PHOTOSHOP	GIMP
Mechanism	Batch command runs the GUI app against a folder	Command-line batch mode runs headless, no GUI at all
Common resize/convert shortcut	Image Processor (dedicated dialog)	No dedicated dialog — written as a script
Scripting language options	N/A (uses recorded Actions)	Script-Fu (Scheme) or Python-Fu (Python)
Runs without a display	No — Batch still drives the visible application	Yes — genuinely headless via `-i -b`

ALWAYS TEST ON A SMALL SUBSET BEFORE RUNNING THE FULL BATCH

Running a batch operation against ten representative files first — rather than the full folder of several hundred — catches mistakes while they're cheap to fix. A wrong resize dimension, a misplaced watermark, or an incorrect output format costs a few seconds to notice and correct on ten files; the same mistake found only after processing all several hundred means redoing all of them.

BATCH PROCESSING AMPLIFIES CHAPTER 6'S SILENT-FAILURE RISK

Chapter 6 named a real limitation: an Action (or a naive script) can fail silently or behave oddly on a file that doesn't match the exact conditions it was built around. Batch processing doesn't just carry that risk forward — it multiplies it, since one file out of several hundred with an unexpected property (a different color mode, an unusually-named layer, a corrupted file) can silently produce a wrong result for that one file, or in some setups halt the entire batch partway through. A production-grade batch script needs explicit error handling — checking assumptions before acting, and logging which files succeeded or failed — which is genuinely more work than the "happy path" script that only handles the expected case.

Hands-On Exercises

EXERCISE 1

Explain the specific mechanical difference between how Photoshop's Batch command runs an Action across a folder and how GIMP's command-line batch mode runs a script across a folder.

View solution

EXERCISE 2

A GIMP user who already knows Python well but has never learned Scheme wants to write a batch script. Using this chapter's own material, explain what option they have, and why it's a genuinely more approachable choice for them specifically.

View solution

EXERCISE 3

A batch script resizes and watermarks 400 product photos overnight. The next morning, three of the 400 output files are blank or wrong, with no error message anywhere. Using this chapter's own warning box, explain why this likely happened and what should have been built into the script to catch it.

View solution

CHAPTER 7 QUICK REFERENCE

- **Photoshop:** File > Automate > Batch runs a recorded Action against a folder; Image Processor is a dedicated shortcut for resize/convert jobs
- **GIMP:** command-line `gimp -i -b '(...)'` runs headless, with no GUI at all — a genuinely different mechanism, not just a different menu
- GIMP offers **Python-Fu** as an alternative to Script-Fu — same PDB, ordinary Python syntax instead of Scheme
- Always test a batch operation on a small representative subset before running the full folder
- Batch processing **multiplies** Chapter 6's silent-failure risk — production scripts need explicit error handling and logging, not just the happy-path sequence

Raster Editing Fundamentals' Chapter 8 covered basic filters applied one at a time. This course's own Chapter 4 covered Smart Filters — a single filter, kept editable, on a Smart Object. This chapter covers what happens once you deliberately combine *several* filters together as a stack: the order they're applied in, and how they interact, becomes its own decision — not just an afterthought of which filters happen to be available.

Why Filter Order Matters

The same set of filters produces genuinely different results depending on the order they're applied in, because each filter operates on whatever the image already looks like at that point in the stack — not on some fixed "original" state. Sharpening an image and then adding film grain produces crisp detail with grain sitting cleanly on top of it. Adding grain first and then sharpening instead sharpens the grain itself, exaggerating it into harsher, more visible noise. Same two filters, same settings, a genuinely different result purely from order — the same underlying principle this site has already met in layer stacking order, object stacking order, and node-graph evaluation order elsewhere.

Photoshop's Filter Gallery

The **Filter Gallery** is a dedicated dialog for building a multi-filter stack with a single live preview: add a filter, adjust its settings, add another on top, reorder any of them by dragging, toggle individual filters off temporarily to compare, all before committing anything. Applied to a Smart Object (Chapter 4), the entire stack built inside the Filter Gallery becomes *one single* Smart Filter entry in the Layers panel — reopening it brings back the full stack, still separately adjustable, exactly as it was left.

GIMP's GEGL-Based Filter Workflow

GIMP has no single dialog equivalent to the Filter Gallery for combining multiple filters into one unified, live-previewed stack. Each GEGL-based filter offers its own live on-canvas preview individually (as covered in Chapter 4), but building a multi-filter effect in GIMP means applying filters one at a time as separate, sequential operations rather than assembling them inside one shared workspace. Each individual filter remains adjustable via GEGL's on-canvas settings *if reopened before any further edit is made* — the same caveat about GEGL's narrower non-destructive scope already named in Chapter 4 applies here too, at a larger scale, across an entire stack rather than one filter.

ASPECT	PHOTOSHOP	GIMP
Multi-filter workspace	Filter Gallery — one dialog, live combined preview	No direct equivalent — filters applied sequentially, one at a time
Reordering a stack	Drag filters within the Filter Gallery	Undo and reapply in a different sequence
Stack stays editable as a whole	Yes, as one Smart Filter entry (on a Smart Object)	Only per-filter, and only before further edits (per Ch.4)

DOCUMENT YOUR FILTER RECIPE FOR CONSISTENCY

A stack that took real trial and error to get right — say, the specific sharpen → grain → vignette order and settings used for a client's product line — is worth writing down explicitly (filter names, order, and settings), not just left as a memory of "what felt right." This matters directly for Chapter 7's batch processing: a documented, repeatable recipe is exactly what a batch script needs to encode, and a recipe that only exists as a vague memory can't be reliably automated or reproduced on the next hundred images.

MORE FILTERS STACKED ISN'T AUTOMATICALLY A BETTER RESULT

Each additional filter compounds both processing time and visual risk — stacking two or three sharpen-family filters, for instance, doesn't multiply sharpness usefully; it tends to produce harsh, artificial-looking edge halos well past the point of looking intentional. The same caution against "more is automatically better" applies here that this site has already met with lighting setups, animation keyframes, and EQ/compression settings elsewhere: a filter stack should be judged by whether the final result looks right, not by how many filters went into producing it.

Hands-On Exercises

EXERCISE 1

Explain why applying a sharpen filter and then a grain filter produces a different result than applying the same two filters in the reverse order.

View solution

EXERCISE 2

Explain what happens in Photoshop if a multi-filter stack was applied inside the Filter Gallery to a Smart Object, and the retoucher later needs to remove just the middle filter from that stack. Then explain how the same change would need to be handled in GIMP.

View solution

EXERCISE 3

A retoucher stacks three separate sharpening filters on a photo, reasoning "if a little sharpening looks good, three passes must look three times as good." Using this chapter's own warning box, explain what's wrong with that reasoning.

View solution

CHAPTER 8 QUICK REFERENCE

- Filter **order** genuinely changes the result — each filter operates on whatever the stack currently looks like, not a fixed original
- **Photoshop:** Filter Gallery combines multiple filters into one live-previewed, reorderable stack — becomes one editable Smart Filter on a Smart Object
- **GIMP:** no equivalent multi-filter workspace — filters applied sequentially, each only reopenable before further edits (per Ch.4's own GEGL caveat)
- Document a working filter recipe explicitly — it's exactly what Chapter 7's batch scripts need to encode reliably
- Stacking more filters isn't automatically better — compounds both processing time and visible artifacts (e.g. sharpen halos)

Chapter 5 mentioned that GIMP's Wavelet Decompose script isn't part of a default install and would need installing separately — a promise this chapter now delivers on. More broadly, both Photoshop and GIMP can be extended well beyond their own built-in feature set with plugins and extensions written by someone other than the core development team. How that ecosystem is structured, and what it costs, is a real, practical difference between the two tools — not a minor footnote.

What a Plugin or Extension Actually Is

A plugin (Photoshop's usual term) or extension/script (GIMP's usual terms) is code, written by a third party, that adds new capability to the base application — a new filter, a new panel, a new automated workflow — installed separately from the application itself rather than being part of what ships out of the box.

Photoshop's Plugin Ecosystem

Photoshop's plugin ecosystem is built around **Adobe Exchange**, a marketplace of third-party plugins and panels built by independent commercial developers on top of Photoshop's own plugin API. Entire categories of professional retouching and workflow tools exist as commercial plugin suites — specialized noise-reduction engines, panel-based batch tools, texture and sky-replacement libraries — most sold as a one-time purchase or ongoing subscription, separate from the Creative Cloud subscription itself. This is a genuine, real cost on top of Photoshop's own subscription, not a hypothetical one.

Installing a GIMP Script: Wavelet Decompose, Concretely

GIMP's installation model for a Script-Fu script is structurally simple, and worth walking through concretely since Chapter 5 left it as a forward reference:

1. Download the script file (typically a plain text file with a `.scm` extension, like Wavelet Decompose).
2. Find GIMP's own scripts folder via **Edit > Preferences > Folders > Scripts** — GIMP lists the exact directory path it scans for scripts.
3. Copy the `.scm` file into that folder.
4. Run **Filters > Script-Fu > Refresh Scripts** to make GIMP pick up the new script immediately — no restart required.

A Python-Fu plugin follows a similar idea but installs into a separate **plug-ins** folder instead of the scripts folder, and on Linux/macOS may additionally need executable permissions set on the file before GIMP will recognize it. Either way, this is a fundamentally different installation model than Photoshop's typical plugin installer — a script is just a text file dropped into the right folder, not a packaged, signed installer application.

GIMP's Free, Community-Maintained Ecosystem

The overwhelming majority of GIMP scripts and plugins — including Wavelet Decompose — are free, written and shared by individual community members through forums, GitHub repositories, and community script collections, reflecting the same free/open-source ethos as GIMP itself. There's no central paid marketplace curating, vetting, or guaranteeing ongoing support for these scripts the way Adobe Exchange does for its commercial listings.

ASPECT	PHOTOSHOP	GIMP
Marketplace	Adobe Exchange — curated, commercial	No central marketplace — forums, GitHub, community collections
Typical cost	Often paid — one-time purchase or subscription	Overwhelmingly free
Installation	Packaged installer application	Copy a script file into a designated folder, then refresh
Ongoing support/compatibility	Vendor-maintained, often part of what the price covers	Community-maintained, no guarantee of upkeep

CHECK COMPATIBILITY BEFORE INSTALLING ANY SCRIPT OR PLUGIN

Always confirm a script or plugin is documented as compatible with your specific application version before installing it. This matters especially for GIMP: when GIMP 2.10 introduced the GEGL engine (Chapter 4), a number of older community Script-Fu scripts, written against the prior engine's internals, stopped working correctly without updates. A script's own documentation or the community page it was shared on is usually the fastest way to confirm it's still actively compatible before relying on it for real work.

"FREE" AND "PAID" CARRY A REAL, HONEST TRADEOFF — NOT A SIMPLE WIN FOR EITHER SIDE

A commercial Photoshop plugin's price is often genuinely paying for something real: vendor-maintained compatibility as Photoshop itself updates, and a support channel if something breaks. A free GIMP script has no equivalent guarantee — if it stops working correctly after a GIMP update (exactly as happened broadly with the GEGL transition), there's no vendor obligated to fix it, only whichever community member originally wrote it, if they're still maintaining it at all. Free is a genuine, real advantage — but it isn't a strictly better deal in every respect; it trades guaranteed vendor support for no cost, rather than offering both at once.

Hands-On Exercises

EXERCISE 1

Walk through, step by step, how you would install the Wavelet Decompose script into GIMP so it appears as a usable filter.

View solution

EXERCISE 2

A colleague says "GIMP's plugin ecosystem is strictly better than Photoshop's, since everything is free." Using this chapter's own warning box, explain what this claim overlooks.

View solution

EXERCISE 3

A studio updates GIMP to a new major version, and a community script they've relied on for years suddenly stops working correctly. Using this chapter's own material, explain why this happened and what could have reduced the risk beforehand.

View solution

CHAPTER 9 QUICK REFERENCE

- **Photoshop:** Adobe Exchange marketplace, largely commercial plugins with vendor-maintained support
- **GIMP:** no central marketplace — scripts shared via forums/GitHub, overwhelmingly free, community-maintained
- Installing a GIMP Script-Fu script: copy the `.scm` file into the Scripts folder (Edit > Preferences > Folders > Scripts), then Filters > Script-Fu > Refresh Scripts
- Python-Fu plugins install into a separate plug-ins folder, and may need executable permissions on Linux/macOS
- Always check a script/plugin's compatibility before installing — GIMP's GEGL transition broke a real number of older scripts
- "Free" and "paid" trade off cost against guaranteed vendor support — neither is a strictly better deal in every respect

Raster Editing Fundamentals' own capstone worked one photo, end to end, by hand — and named this course's own capstone directly as where that exact discipline scales up to a real production batch, automated via scripting. This capstone does exactly that: build one proven retouching recipe on a single reference photo, then automate it across an entire batch of product photos, using every technique from Chapters 2 through 9.

The Scenario

A client has delivered 60 raw product photos, each shot against a cluttered studio background, needing: the product masked out cleanly (including some fine, hair-like fringe on a fabric sample), composited onto the client's own consistent branded background, surface texture retouched to remove shipping-handling marks without losing real material texture, a subtle branded filter look applied consistently, and every file watermarked and exported — with all 60 files needing to go out the same day.

Step 1 — Building the Mask on a Reference Photo (Chapter 2)

Working on one representative photo first, build a precise channel-based mask around the product, using whichever color channel shows the strongest contrast against the studio background, refined enough to hold up against the fabric sample's own fine fringe detail.

Step 2 — Compositing Onto the Branded Background (Chapter 3)

Composite the masked product onto the client's branded background, matching color and lighting direction between the two, and feathering the mask's edge slightly so the fringe detail preserved in Step 1 reads as naturally photographed rather than pasted on.

Step 3 — Non-Destructive Retouching and the Brand's Filter Look (Chapters 4 & 8)

Convert the composited product layer to a Smart Object before applying the brand's signature filter stack — a specific, previously agreed sharpen-then-subtle-vignette combination — so the whole stack stays reopenable and adjustable per photo if the client requests a tweak later, rather than being baked in destructively.

Step 4 — Frequency-Separation Cleanup (Chapters 5 & 9)

Remove the shipping-handling marks from the product's surface using frequency separation, correcting tone on the Low Frequency layer without touching the real material texture on the High Frequency layer. In GIMP, this step uses the Wavelet Decompose script installed back in Chapter 9 to build the frequency layers, rather than assembling them by hand each time.

Step 5 — Recording the Recipe (Chapter 6)

With the full sequence proven on this one reference photo, record it as a Photoshop Action, or write it as a GIMP script calling the same sequence of PDB operations — turning a manually-proven workflow into something repeatable without re-performing every step by hand on each of the 60 files.

Step 6 — Running the Batch (Chapter 7)

Test the recorded Action or script against five representative files first, per Chapter 7's own tip — confirming the mask, composite, filter stack, and frequency-separation cleanup all hold up correctly before committing to the full run. Then run the Batch command (or the equivalent headless GIMP command-line invocation) across the remaining files.

Step 7 — Catching the Files That Didn't Match Assumptions (Chapters 6 & 7)

Two of the 60 files were shot at a different angle than the rest, and the recorded recipe's masking step doesn't hold up cleanly on them — exactly the silent-failure risk both Chapter 6 and Chapter 7 warned about. Because the batch run was logged, these two files are caught and flagged rather than shipped incorrectly, and are corrected individually by hand before final delivery.

Step 8 — Final Export

Export all 60 finished files — the corrected two included — to the client's required format and naming convention, with the fully-layered working files for each preserved separately in case of a future revision request.

CAPSTONE STEP	CHAPTER IT DRAWS FROM
Step 1 — Building the mask	Chapter 2
Step 2 — Compositing onto the background	Chapter 3
Step 3 — Smart Object + brand filter stack	Chapters 4 & 8
Step 4 — Frequency-separation cleanup	Chapters 5 & 9
Step 5 — Recording the recipe	Chapter 6
Step 6 — Running the batch	Chapter 7
Step 7 — Catching mismatched files	Chapters 6 & 7
Step 8 — Final export	Raster Editing Fundamentals, Chapter 9

THE WHOLE COURSE IS REALLY ONE LESSON, APPLIED AT INCREASING SCALE

Every chapter in this course took one part of Raster Editing Fundamentals' own discipline and pushed it further — one mask into a precise channel-based one, one blend into a full composite, one filter into a reopenable stack, one retouch into frequency separation, and finally one manual workflow into an automated, batch-scale one. This capstone doesn't introduce anything new; it's the same "make it look right, make it repeatable" idea this whole two-course track has followed from its very first chapter, now running across sixty files instead of one.

AUTOMATION DOESN'T REMOVE THE NEED FOR HUMAN JUDGMENT

Two files out of sixty still needed a human to notice they didn't fit the recorded recipe's assumptions, and to fix them individually. This isn't a failure of automation — it's exactly what Chapters 6 and 7 predicted would eventually happen at real production scale, and precisely why testing on a representative subset first and logging the full run matters. Automation handles the repeatable 95%+ reliably; a retoucher's own judgment is still what catches and corrects the rest.

Hands-On Exercises

EXERCISE 1

Explain why this capstone builds and proves the entire recipe on one reference photo (Steps 1–4) before recording or scripting anything (Step 5), rather than starting directly with automation.

View solution

EXERCISE 2

Explain why converting the product layer to a Smart Object in Step 3, before applying the brand's filter stack, specifically matters for a batch of 60 files rather than just one.

View solution

EXERCISE 3

Explain how this capstone's own outcome — 58 files processed correctly, 2 caught and fixed by hand — illustrates the relationship between automation and human judgment described in this chapter's own warning box, rather than treating either the automation or the two manual fixes as a failure.

View solution

COURSE COMPLETE

Advanced Compositing & Retouching — 10 of 10 chapters complete. Combined with Raster Editing Fundamentals' own 10 chapters, the full Photoshop & GIMP track — 20 chapters in total — is now complete.

CHAPTER 10 QUICK REFERENCE

- Build and prove the full recipe on **one reference photo** before automating anything
- Every technique from Chapters 2–9 (masking, compositing, Smart Objects, frequency separation, scripting, batch processing, filter stacking, plugins) is used together in one workflow
- Test the recorded Action/script on a small subset before running the full batch
- Automation handles the repeatable majority reliably — human judgment is still needed to catch and fix the files that don't fit its assumptions
- This closes the full **Photoshop & GIMP** track — Raster Editing Fundamentals (10 ch.) + Advanced Compositing & Retouching (10 ch.) = 20 chapters